UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A209 926

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Ada Compiler Validation Summary Report: Gould, Inc APLEX Ada Compiler, Revision 2.2, Gould/Encore PowerNode Model 9080 (Host) to Gould/Encore CONCEPT/32 Model 6744 (Target), 890418W1.10070

5. TYPE OF REPORT & PERIOD COVERED
18 Apr. 1989 to 18 Apr. 1990

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Wright-Patterson AFB
Dayton, OH, USA

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS
Wright-Patterson AFB
Dayton, OH, USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Wright-Patterson AFB
Dayton, OH, USA

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)
Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Gould, Inc., APLEX Ada Compiler, Revision 2.2, Wright-Patterson AFB, Gould/Encore PowerNode Model 9080 under UTX/32, Revision 2.1 (Host) to Gould/Encore CONCEPT/32 Model 6744 under Ada Real Time Executive (bare machine), ACVC 1.10.

89 6 29 174

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada Compiler Validation Summary Report:

Compiler Name: APLEX Ada Compiler, Revision 2.2

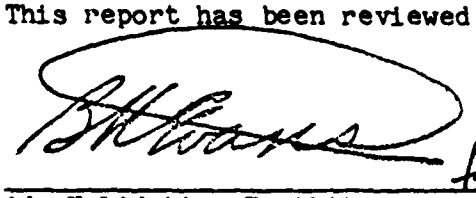
Certificate Number: 890418W1.10070

Host: Gould/Encore PowerNode Model 9080 under UTX/32, Revision 2.1

Target: Gould/Encore CONCEPT/32 Model 6744 under Ada Real Time Executive (bare machine)

Testing Completed 18 April 1989 Using ACVC 1.10

This report has been reviewed and is approved.



Ada Validation Facility
Steve P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director, AJPO
Department of Defense
Washington DC 20301

AVF Control Number: AVF-VSR-257.0589
88-11-16-GOU

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890418W1.10070
Gould, Inc.
APLEX Ada Compiler, Revision 2.2
Gould/Encore PowerNode Model 9080 Host and
Gould/Encore CONCEPT/32 Model 6744 Target

Completion of On-Site Testing:
18 April 1989

Prepared By:
Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: APLEX Ada Compiler, Revision 2.2

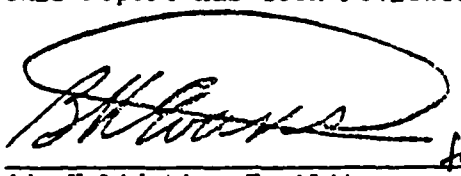
Certificate Number: 890418W1.10070

Host: Gould/Encore PowerNode Model 9080 under UTX/32, Revision 2.1

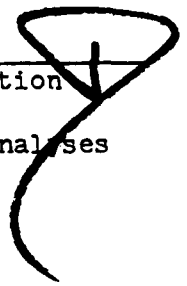
Target: Gould/Encore CONCEPT/32 Model 6744 under Ada Real Time Executive (bare machine)

Testing Completed 18 April 1989 Using ACVC 1.10

This report has been reviewed and is approved.



Ada Validation Facility
Steve P. Wilson
Technical Director
ASD/SCEL
Wright-Patterson AFB OH 45433-6503



Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director, AJPO
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 18 April 1989 at Fort Lauderdale FL.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCEL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including

cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — An ACVC test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every

illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate

tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: APLEX Ada Compiler, Revision 2.2

ACVC Version: 1.10

Certificate Number: 890418W1.10070

Host Computer:

Machine:	Gould/Encore PowerNode Model 9080
Operating System:	UTX/32 Revision 2.1
Memory Size:	16 megabytes

Target Computer:

Machine:	
Board:	Gould/Encore CONCEPT/32 Model 6744
CPU:	6744
Bus:	SELBUS
I/O:	SELBUS
Timer:	N/A
Operating System:	(bare machine)
Memory Size:	8 megabytes

Communications Network: Ethernet

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, LONG_INTEGER, and LONG_FLOAT in package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) Some of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses no extra bits for extra range. (See test C35903A.)

(4) No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) No exception is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is not gradual. (See tests C45524A..Z.)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z.)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z.)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

(2) NUMERIC_ERROR is raised when a null array type with INTEGER'LAST + 2 components is declared. (See test C36202A.)

(3) NUMERIC_ERROR is raised when a null array type with SYSTEM.MAX_INT + 2 components is declared. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is not supported for non-library units. (See tests LA3004A..B, EA3004C..D, and CA3004E..F.)

i. Generics

(1) Generic unit declarations, bodies, and subunits can be compiled in separate compilations. (See tests CA1012A and CA3011A.)

(2) If a generic unit body or one of its subunits is compiled or recompiled after the generic unit is instantiated, the unit instantiating the generic is made obsolete. The obsolescence is recognized at binding time, and the binding is stopped. (See tests CA2009C, CA2009F, BC3204C, and BC3205D.)

j. Input and output

(1) The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

(2) The package DIRECT_IO cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

(3) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E, CE2102N, and CE2102P.)

(4) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J, CE2102R, CE2102T, and CE2102V.)

(5) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K.)

(6) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(7) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(8) RESET and DELETE operations are supported for text files. (See tests CE3102F..G, CE3104C, CE3110A, and CE3114A.)

(9) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(10) Temporary sequential files are given names and not deleted when closed. (See test CE2108A.)

(11) Temporary direct files are given names and not deleted when closed. (See test CE2108C.)

(12) Temporary text files are given names and not deleted when closed. (See test CE3112A.)

(13) Only one internal file can be associated with each external file for sequential files. (See tests CE2107A..E, CE2102L, CE2110B, and CE2111D.)

(14) Only one internal file can be associated with each external file for direct files. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(15) Only one internal file can be associated with each external file for text files. (See tests CE3111A..E, CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 43 tests had been withdrawn because of test errors. The AVF determined that 331 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 215 executable tests that use floating-point precision exceeding that supported by the implementation . Modifications to the code, processing, or grading for eleven tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	127	1132	1999	17	23	45	3343
Inapplicable	2	6	317	0	5	1	331
Withdrawn	1	2	34	0	6	0	43
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	197	568	547	247	172	99	161	333	132	36	250	326	275	3343
Inappl	15	81	133	1	0	0	5	0	5	0	2	43	46	331
Wdrn	1	1	0	0	0	0	0	1	0	0	1	35	4	43
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 43 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110	ED7004B
ED7005C	ED7005D	ED7006C	ED7006D	CD7105A	CD7203B
CD7204B	CD7205C	CD7205D	CE2107I	CE3111C	CE3301A
CE3411B					

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 331 tests were inapplicable for the reasons indicated:

a. The following 215 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113K..Y	C35705K..Y	C35706K..Y	C35707K..Y
C35708K..Y	C35802K..Z	C45241K..Y	C45321K..Y
C45421K..Y	C45521K..Z	C45524K..Z	C45621K..Z
C45641K..Y	C46012K..Z		

b. C35508I, C35508J, C35508M, and C35508N are not applicable because this implementation does not support enumeration representation clauses for BOOLEAN types.

c. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

d. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

e. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 47.

f. C52008B is not applicable because this implementation does not support a record type with four discriminants of type integer having default values. The size of this object exceeds the maximum object size of this implementation and NUMERIC_ERROR is raised.

g. C86001F is not applicable because, for this implementation, the package TEXT_IO is dependent upon package SYSTEM. These tests recompile package SYSTEM, making package TEXT_IO, and hence package REPORT, obsolete.

h. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

i. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

j. CA2009C, CA2009F, BC3204C, and BC3205D are not applicable because they contain instantiations of generics in cases where the body is not available at the time of the instantiation. As allowed by AI-00408/07 this compiler creates a dependency on the missing body so that when the actual body is compiled, the unit containing the instantiation becomes obsolete.

k. LA3004B, EA3004D, and CA3004F are not applicable because this implementation does not support pragma INLINE for non-library units.

l. CD1009C, CD2A41A..B (2 tests), CD2A41E, and CD2A42A..J (10 tests) are not applicable because this implementation does not support size clauses for floating point types using less than 32 bits.

m. CD2A52A..D (4 tests), CD2A52G..J (4 tests), CD2A54A..D (4 tests), and CD2A54G..J (4 tests) are not applicable because this implementation does not support fixed point types using less than 16 bits.

n. CD2A61I and CD2A61J are not applicable because this implementation does not support size clauses for array types, which imply compression, with component types of composite or floating point types.

o. CD2A84B..I (8 tests) and CD2A84K..L (2 tests) are not applicable because this implementation does not support size clauses for access types using less than 32 bits.

p. AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

q. AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.

r. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

s. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

t. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

u. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

v. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

w. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

x. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

y. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

z. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

aa. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

ab. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

ac. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

ad. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

ae. CE2102V is inapplicable because this implementation supports open with

OUT_FILE mode for DIRECT_IO.

af. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ag. CE3102E is inapplicable because this implementation supports CREATE with IN_FILE mode for text files.

ah. CE3102F is inapplicable because this implementation supports RESET for text files.

ai. CE3102G is inapplicable because this implementation supports deletion of an external file for text files.

aj. CE3102I is inapplicable because this implementation supports CREATE with OUT_FILE mode for text files.

ak. CE3102J is inapplicable because this implementation supports OPEN with IN_FILE mode for text files.

al. CE3102K is inapplicable because this implementation supports OPEN with OUT_FILE mode for text files.

am. CE2107A..E (5 tests), CE2107L, CE2110B, and CE2111D are not applicable because multiple internal files cannot be associated with the same external file for sequential files. The proper exception is raised when multiple access is attempted.

an. CE2107F..H (3 tests), CE2110D, and CE2111H are not applicable because multiple internal files cannot be associated with the same external file for direct files. The proper exception is raised when multiple access is attempted.

ao. CE3111A..B (2 tests), CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file for text files. The proper exception is raised when multiple access is attempted.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for eleven tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

BA3006A BA3006B BA3007B BA3008A BA3008B BA3013A

At the recommendation of the AVO, the following modifications were made to compensate for legitimate implementation behavior.

a. "TTYPE'STORAGE_SIZE" in CD2C11A..B was changed to 3072 since the original value of 1024 was insufficient for this implementation.

b. In tests CE2108B, CE2108D, and CE3112B, the lines which check whether temporary files can be created were commented out because of the way in which temporary file names are constructed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the APLEX Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the APLEX Ada Compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	Gould/Encore PowerNode Model 9080
Host operating system:	UTX/32, Revision 2.1
Target computer:	Gould/Encore CONCEPT/32 Model 6744
Target operating system:	Ada Real Time Executive (bare machine)
Compiler:	APLEX Ada Compiler, Revision 2.2

The host and target computers were linked via Ethernet.

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the Gould/Encore CONCEPT/32 Model 6744. Results were printed from the from the host computer.

The compiler was tested using command scripts provided by Gould, Inc. and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

OPTION	EFFECT
-i	Enable optimizer and process pragma include.
-c	Produce a compilation listing.

Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Fort Lauderdale FL and was completed on 18 April 1989.

Power failures during the validation caused some system down time. The host was not dedicated to the validation.

APPENDIX A

DECLARATION OF CONFORMANCE

Gould, Inc. has submitted the following Declaration of Conformance concerning the APLEX Ada Compiler.

DECLARATION OF CONFORMANCE

Compiler Implementor: Gould/Computer Systems Inc.
Ada Validation Facility: ASD/SIOL, Wright-Patterson AFB, OH
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name:	APLEX™ Ada Compiler	Revision 2.2
Host Architecture ISA:	Gould/Encore PowerNode Model 9080	
OS& VER #:	UTX/32	Revision 2.1
Target Architecture ISA:	Gould/Encore CONCEPT/32 Model 6744	
OS&VER #:	Ada Real Time Executive (Bare machine)	

Derived Compiler Registration

Derived Compiler Name:	APLEX™ Ada Compiler	Revision 2.2
Host Architecture ISA:	Gould/Encore PowerNode Model 90XX, 60XX CONCEPT/32 Model 67XX	
OS&VER #:	UTX/32	Revision 2.1
Target Architecture ISA:	Gould/Encore PowerNode Model 90XX, 60XX CONCEPT/32 Model 97XX, 67XX	
OS&VER #:	Ada Real Time Executive (Bare machine)	

Implementor's Declaration

I, the undersigned, representing Gould/Computer Systems Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that Gould/Computer Systems Inc. is the owner of record of the Ada language compiler listed above, and as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

Mary F. Macomber Date: 4/21/89

Gould/Computer Systems Inc.
Mary F. Macomber, Senior Manager, Major Corporate Agreements

Owner's Declaration

I, the undersigned, representing Gould/Computer Systems Inc., take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A. I have reviewed the Validation Summary Report for the compiler and concur with the contents. I also affirm that the CONCEPT/32 computer architectures listed (97XX and 67XX models) herein are of equivalent architecture to the CONCEPT/32 as described in the documentation which was submitted with our 1.8 validations. I also affirm that the PowerNode computer architectures listed (60XX and 90XX models) herein are of equivalent architecture to the PowerNode 9080 as described in the documentation which was submitted with our 1.8 validations.

Mary F. Macomber Date: 4/21/89

Gould/Computer Systems Inc.
Mary F. Macomber, Senior Manager, Major Corporate Agreements

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the APLEX Ada Compiler, Revision 2.2, as described in this Appendix, are provided by Gould, Inc.. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...

   type INTEGER is range -(2**31) .. (2**31)-1;
   type SHORT_INTEGER is range -(2**15) .. (2**15)-1;
   type LONG_INTEGER is range -(2**63) .. (2**63)-1;

   type FLOAT is digits 6 range -7.23698E+75 .. 7.23698E+75;
   type LONG_FLOAT is
        digits 14 range -7.2370055773320E+75 .. 7.2370055773320E+75;

   type DURATION is delta 2#1.0#E-14 range -86400.0 .. 86400.0;

   ...

end STANDARD;
```

Appendix F.
Implementation-Dependent Characteristics

1 The Ada language definition allows for certain machine-dependencies in a controlled manner. No machine-dependent syntax or semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementation-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in Chapter 13, and certain allowed restrictions on representation clauses.

2 The reference manual of each Ada implementation must include an appendix (called Appendix F) that describes all implementation-dependent characteristics. The Appendix F for a given implementation must list in particular:

3 (1) The form, allowed places, and effect of every implementation-dependent pragma.

4 (2) The name and the type of every implementation-dependent attribute.

5 (3) The specification of the package SYSTEM (see 13.7).

6 (4) The list of all restrictions on representation clauses (see 13.1).

7 (5) The conventions used for any implementation-generated name denoting implementation-dependent components (see 13.4).

8 (6) The interpretation of expressions that appear in address clauses, including those for interrupts (see 13.5).

9 (7) Any restriction on unchecked conversions (see 13.19.2).

10 (8) Any implementation-dependent characteristics of the input-output packages (see 14).

1.6 The compiler detects all errors defined in this section. Unless the error is fatal, compilation does not stop. However, any error inhibits the generation of object code.

Error messages contain:

- The line number in the source code where the error was detected.
- A copy of the erroneous source line with a marker showing the position of the error.
- Descriptive text indicating the meaning of the error.
- Any additional information which more precisely identifies the cause of the error (i.e., a symbol or the section number in the LRM that applies to the source error.)

When an error is detected, the compiler attempts reasonable recovery actions so syntax and semantic checking can continue. For example, it may substitute a seemingly reasonable construct for the one in error. The construct may or may not be the intended one. If a compilation is aborted, the compiler closes all files that were opened and accessed during the compilation.

Undeclared identifiers are considered to be errors.

Information is not generated for use in resolving ambiguous overloads.

No warning is produced under the following circumstances:

- A construct consumes a large amount of memory or takes a long time to execute.
- The result of a real expression has less than 1 significant digit.
- A statement is unreachable.
- Declared entities are not used.
- Variables are uninitialized.
- Loops are endless.

A warning is issued for a statement whose static properties guarantee an exception will be raised.

Warnings are issued for ignored and unsupported pragmas.

Chapter 2. Lexical Elements

2.1 The host and target character set is the ASCII character set.

2.2 The compiler accepts as many as 199 characters on a source line.

Chapter 3. Declarations and Types

3.2.1 The compiler does not produce warning messages for uninitialized variables and will not reject a program for this reason.

3.5.1 The maximum number of elements in an enumeration type is 1000.

3.5.5 The range for standard integers is:

```
INTEGER'FIRST = -(2**31)
INTEGER'LAST = (2**31)-1
INTEGER'WIDTH = 11
INTEGER'SIZE = 32

LONG_INTEGER'FIRST = -(2**63)
LONG_INTEGER'LAST = (2**63)-1
LONG_INTEGER'WIDTH = 20
LONG_INTEGER'SIZE = 64

SHORT_INTEGER'FIRST = -(2**15)
SHORT_INTEGER'LAST = (2**15)-1
SHORT_INTEGER'WIDTH = 6
SHORT_INTEGER'SIZE = 16
```

Refer to type CHARACTER in Annex C for the image of a character. Images of non-graphic characters are as shown in paragraph 13 of Annex C.

3.5.8 The attributes of floating-point numbers are:

```
FLOAT'DIGITS = 6
FLOAT'SAFE_EMAX = 252
FLOAT'MACHINE_EMAX = 63
FLOAT'MACHINE_EMIN = -64
FLOAT'MACHINE_MANTISSA = 6
FLOAT'MACHINE_OVERFLOWS = TRUE
FLOAT'MACHINE_RADIX = 16
FLOAT'MACHINE_ROUNDS = FALSE
FLOAT'SIZE = 32
```

LONG_FLOAT'DIGITS = 14
LONG_FLOAT'SAFE_EMAX = 252
LONG_FLOAT'MACHINE_EMAX = 63
LONG_FLOAT'MACHINE_EMIN = -64
LONG_FLOAT'MACHINE_MANTISSA = 13
LONG_FLOAT'MACHINE_OVERFLOWS = TRUE
LONG_FLOAT'MACHINE_RADIX = 16
LONG_FLOAT'MACHINE_ROUNDS = FALSE
LONG_FLOAT'SIZE = 64

The type SHORT_FLOAT is not supported at this time.

3.5.9 Fixed point types are represented as integers with an implied binary point.

Chapter 4. Names and Expressions

4.8 Pragma CONTROLLED is not supported.

The storage space occupied by an object created by an allocator is not reclaimed when the object becomes inaccessible, except when storage by collection is used via the STORAGE_SIZE representation specification.

4.10 There is no limit on the range of literal values or on the accuracy of real literal expressions. Real literal expressions are computed using an arbitrary precision universal arithmetic package.

Chapter 6. Subprograms

6.3.2 Pragma INLINE is supported except for subprograms which are library units, that is, compilation units with no enclosing scope such as a main procedure.

Chapter 9. Tasks

9.2 Task space is acquired from the heap upon task allocation. Task space is released when the task is terminated.

9.3 On UTX/32 and MPX-32 targets, tasks are scheduled only when a scheduling event occurs. The expiration of a delay does not cause a scheduling event. Thus, it is possible for a low priority task to dominate the processor, even if a higher priority task is ready to run.

On the Ada Real Time Executive, task scheduling occurs on the expiration of a delay and the completion or initiation of an input/output call. Thus, the task that is currently running is always the highest priority task that is ready to run.

A scheduling event occurs if the currently active task executes an entry call, accept statement, delay statement, select statement, or abort statement, or if it elaborates another task or terminates.

9.4 A task that is initiated in an imported library unit continues to execute until it completes.

9.5 Storage used by a task is delivered to the total pool of storage when the task terminates.

9.6 The values for type DURATION are:

```
DURATION'DELTA = 1.0/(2.0**14)
DURATION'FIRST = -86400.0
DURATION'LAST = 86400.0
```

9.8 The values for type PRIORITY are:

```
PRIORITY'FIRST = 0
PRIORITY'LAST = 255
```

The default is 128.

9.11 Pragma SHARED is supported for scalar and access types.

Chapter 10. Program Structure and Compilation Issues

10.1 All main programs must be procedures without parameters.

10.3 Pragma INLINE is supported except for subprograms which are library units, i.e., compilation units with no enclosing scope.

A generic declaration and its corresponding body (and all subunits) need not be part of the same compilation.

Separate compilation of generic specifications, bodies and subunits is allowed from separate files.

10.4 Library management tools provide the ability to create a list of compilation units made obsolete by the compilation of a unit.

Chapter 11. Exceptions

11.7 Pragma SUPPRESS does not affect checks for integer and floating point overflow, division by zero, and task elaboration.

Chapter 13. Representation Clauses and Implementation-Dependent Features

13.1 Pragma PACK is supported.

In a packed record, all spare space between scalar fields is removed. Each scalar field is placed in the next available bit position and allocated the minimum number of bits necessary. It is possible, for example, for an integer field to overlap byte and word boundaries.

Fields that are composite types preserve the size and alignment characteristics of the composite type. Thus, if a field in a packed record is of a type that has a 16-bit alignment, the field will be aligned on a 16-bit boundary within the packed record. This will also have the effect of requiring at least a 16-bit alignment of the packed record.

When arrays are packed, scalar elements are compressed to occupy the minimum necessary number of bits that is a power of 2. Thus, Booleans, integer ranges of 0..1, and enumeration types of two elements are allocated a single bit in a packed array. Integer ranges of 0..3 and enumerations with as many as four elements are packed into two bits. Integer ranges of up to 0..15 and enumerations of up to 16 elements are packed into four bits. Packing has no effect on any other scalars. That is, types that will fit into a byte are allocated one byte per element, those that will fit into 16 bits are allocated a halfword, 32-bit types are allocated a word, and 64-bit types are allocated a doubleword.

Packing an array of a record or array type preserves the size and alignment characteristics of the component type, just as in a record.

13.2 The length clause for the size of a subtype is supported.

The length clause for collection size is supported. The presence of a length clause for an access type implies that storage for that type is to be managed by the collection. The absence of such a clause implies the default storage management.

The length clause for task activation is supported.

The length clause for small of a fixed point type is supported for values of small which are either integers or reciprocals of integers.

13.3 The enumeration representation clause is supported with the restriction that it cannot be used for the predefined type BOOLEAN. Thus, the user cannot change the representations of the predefined constants TRUE and FALSE.

13.4 Record representation clauses are supported. The maximum significant value for the expression following "at mod" is 8. Thus the compiler does not support alignment to boundaries greater than a double word.

Only those records having components that are all statically sized may be the subject of a representation specification. Records containing dynamically sized components may not be the subjects. A base type may not be mapped (via a component clause) into an area larger than it normally occupies; for example, an integer may not be mapped into an area greater than 32 bits. A composite type used as a component of a representation specified record must be aligned so as to preserve the alignment of its constituent components. If a composite type (a record or an array) is used as a field or element in a second composite type, then any representation specification on the second type must retain the bit-alignment of the original type. This explains why the type string (which is a composite type -- an array of characters) cannot be aligned on anything but its original alignment, for example,

```
type integerarray is array (integer range 1..2) of integer;
type integerrecord is
    record
      field : integerarray;
    end record;

for integerrecord use
    record
      field1 at 0 range 2..65;
    end record:
```

This example is illegal because the composite type *integerarray* is aligned on a zero ("0") bit boundary by default and this alignment must be preserved in later uses of this type. The size in bits of a representation specified record with 25 bits actually occupies 32 bits; if the record is used as a component of another representation specified record, 32 bits must be reserved for it.

13.5 Address clauses for objects are supported.

Address clauses for subprograms and packages are not supported.

13.5.1 Interrupts are supported as address clauses for entries for Ada Real Time Executive targets only.

13.6.1 Change of representation is supported.

13.7 The Gould specification for package SYSTEM is:

```
package SYSTEM  is

        type ADDRESS is private;
        type NAME is (Gould_Ada);
        SYSTEM_NAME : constant NAME := Gould_Ada;

        STORAGE_UNIT : constant := 8;
        MEMORY_SIZE : constant := 2**24-1; -- for CONCEPT/PowerNode
          -- or
        MEMORY_SIZE : constant := 2**31-1;  -- for NPL

        --System-Dependent Declarations

        subtype BYTE is INTEGER range 0 .. 2**8-1;
        subtype INTEGER_16 is INTEGER range -2**15 .. 2**15-1;
        subtype INTEGER_32 is INTEGER; -- range -2**31 .. 2**31-1;
        subtype INTEGER_64 is LONG_INTEGER;
                                        -- range -2**63 .. 2**63-1;


        --System-Dependent Named Numbers

        MIN_INT : constant := -(2**63);
        MAX_INT : constant := (2**63)-1;
        MAX_DIGITS: constant := 14;
```

```
    MAX_MANTISSA: constant := 31;
    FINE_DELTA: constant :=  1.0/(2.0**MAX_MANTISSA);
    TICK: constant := 1.0/60.0;

    --Other System Dependent Declarations

    MAX_OBJECT_SIZE : constant := MAX_INT;
    MAX_RECORD_COUNT : constant := MAX_INT;
    MAX_TEXT_IO_COUNT: constant := MAX_INT-1;
    MAX_TEXT_IO_FIELD: constant := 1000;

    subtype PRIORITY is INTEGER range 0 .. 255;

    NULL_ADDRESS : constant ADDRESS;

private

    type ADDRESS is new INTEGER_32;
    NULL_ADDRESS : constant ADDRESS := 0;

end SYSTEM;
```

13.7 Pragmas SYSTEM_NAME, STORAGE_UNIT and MEMORY_SIZE allow only 1 valid value. Thus a recompilation of package SYSTEM will not be performed.

13.7.1 For system-dependent named numbers, the following values apply:

MIN_INT	-2**63
MAX_INT	(2**63)-1
MAX_DIGITS	14
MAX_MANTISSA	31
FINE_DELTA	1.0/(2.0**MAX_MANTISSA)
TICK	1.0/60.0

There are no system generated names for system-dependent components.

13.7.2 The representation attribute 'address is not supported for packages and labels. An 'address attribute for a subprogram refers to the address of the procedure header of the body of the subprogram. For a task it is the address of the first byte of the task's code. It is only implemented for tasks that are declared through the use of task types. For a constant, 'address yields either the address of a constant or NULL_ADDRESS if no storage location is associated with the constant.

13.7.3 For any floating point type FLT, the following attribute values apply:

Attribute	Value
FLT'MACHINE_ROUNDS	FALSE
FLT'MACHINE_RADIX	16
FLT'MACHINE_MANTISSA	(See below)
FLT'MACHINE_EMIN	-64
FLT'MACHINE_EMAX	+63
FLT'MACHINE_OVERFLOWS	TRUE

For floating point types of 6 or fewer decimal digits:

FLT'MACHINE_MANTISSA = 6

For other floating point types:

FLT'MACHINE_MANTISSA = 13

13.8 Machine code insertions are not supported.

13.9 The pragma INTERFACE is supported for Assembler (asmx32), and all Gould Common Languages (FORTRAN, Pascal, and C) on UTX/32, MPX-32, and Ada Real Time Executive targets. Pragma INTERFACE is also supported for cc and F77 on UTX/32 targets.

13.10.2 Unchecked conversions are allowed between variables of types (or subtypes) T1 and T2 provided that:

- they have the same static size
- they are not unconstrained array types

Chapter 14. Input-Output

14.1 When the main program exits, any open files are closed by the system.

Under MPX-32 and ARTE, multiple internal files cannot be opened to the same external file. If an attempt is made to open two internal files to the same external file, the exception USE_ERROR is raised.

Under UTX/32, multiple internal files can only be opened to the same external file for reading.

Calling CREATE with the name of an existing external file does not raise an exception; the old file is overwritten.

14.2 Type COUNT is range 0 .. MAX_INT-1 in TEXT_IO. Type COUNT is range 0 .. MAX_INT in DIRECT_IO.

Instantiation of DIRECT_IO and SEQUENTIAL_IO for unconstrained record and array types is not supported.

14.2.1 *FORM* String Interpretation Conventions for *CREATE* and *OPEN* are as follow:

The *form* string is case insensitive. The exception USE_ERROR is raised if the *form* string is incorrect. Strings may be upper or lower case.

For UTX/32: exclusive

Use of this string causes the file to be opened with a UTX/32 advisory lock.

For MPX-32:

Form string parameters must be separated by a blank. The parameter value immediately follows the equal sign with no intervening blanks. The defaults for file creation are the same as the MPX-32 defaults.

CREATE File:

BLOCKED => {TRUE, FALSE}
TRUE - file data initially blocked (default)
FALSE - file data initially unblocked

SHARED => {TRUE, FALSE}
TRUE - resource is sharable (default)
FALSE - resource is not sharable

ZERO => {TRUE, FALSE}
TRUE - file is zeroed on creation/expansion
FALSE - file is not zeroed on creation/expansion (default)

FILETYPE => {00 .. FF}
Resource type is equivalent to the file type code interpreted as two hexadecimal digits. Valid values are 00 through FF (default value is 00).

MAXINC => n
n size in blocks of each automatic increment for file extension (default is 64 blocks)

MININC => n
n minimum acceptable automatic increment size in blocks if MAXINC cannot be obtained (default value is 32 blocks)

SIZE => n
n initial block allocation size of the file (default is 16 blocks for MPX-32 and 0 blocks for UTX/32 and Ada Real Time Executive targets)

OPEN file:

BLOCKED => {TRUE, FALSE}
TRUE - open in blocked mode (default)
FALSE - open in unblocked mode

SHARED => {TRUE, FALSE}
TRUE - open for explicit shared use
FALSE - open for exclusive use

If SHARED is not specified, the file is opened for implicit shared use (default).

Note: This option is applicable only if the file was created as a shareable file.

PAGE => {TRUE, FALSE}
TRUE - page terminal output
FALSE - do not page output

For direct access, the default size of the created file in UTX/32 equals 0 and in MPX-32 equals 16 blocks.

For ARTE:

When a file is opened, the ARTE file system support determines whether the file resides on a UTX/32 partition or a DFS partition by reading the first path element. The possibilities are:

- The root file system of the boot disk
- Another UTX/32 file system
- A direct file system

The **create, open, close, read, write,** and **delete** procedure calls in the Ada application program correspond to those procedures documented in the Ada LRM. On the **create** or **open** (with implied **create**) statement, the **form** parameter is an implementation specific string parameter that gives more information about the file. The string contains a sequence of DFS parameter names and values. The syntax of the **form** string is:

form : *constant string* := "*param1=value param2=value*";

Parameters for DFS files are listed below. All parameters are optional and can be omitted if the default is to be used.

size specifies the starting file size in tracks. The minimum size is one. The maximum size is limited by the contiguous space available in the file system. (On a UDP, for example, one track is 16 sectors; one sector is 1024 bytes.) The default value is one track.

extent specifies the size of each additional extent (in tracks) if the file must be extended due to passing end of file (EOF) on a write. The minimum extent size is one track. The maximum size is limited by the amount of contiguous file space available in the file system. The default value is one. Use_Error is raised at runtime if the file system support attempts to extend the file and too large a number was specified.

gap specifies the maximum number of cylinders that can separate extents of a file. Gap=0, which is the default, means space can be

allocated in any cylinder. A gap of one or more indicates the new extent must begin in a cylinder no more than that number of cylinders away from the last sector of the previous extent. Smaller numbers mean less seek time when skipping from one extent to the next. The maximum gap size is limited only by the distance between the end of the current extent and the end of the partition.

Use_Error is raised if an attempt to extend a file fails because of a lack of space, or no space is available within the specified gap parameter.

nobuf specifies that I/O is to be done directly from and to the object specified in the user call. Normally I/O is buffered into a holding area in the ARTE Runtime Library and the data is then moved to the user area. This parameter provides an advantage for large data transfers, i.e., greater than 8192 bytes (the internal buffer size). The disadvantage is the possibility of wasted disk space. All unbuffered transfers begin on a disk sector boundary. If a record happens to equal N sectors plus 1 byte, then nearly an entire sector will be wasted for each record that is written. Data should be structured to minimize this loss if this parameter is used. The nobuf parameter does not have a value; its presence in the **form** string causes it to be true; otherwise, it is false.

14.3 For UTX/32, the default standard input and output files are UNIX standard in and standard out. These can be redirected in the usual way. For MPX-32, the default standard input and output files are logical file code SYC for standard input and SLO for standard output. These files are automatically assigned as appropriate for interactive, batch and real time tasks.

For both UTX/32 and MPX-32, the file and line terminators are the normal end of file and end of line. The page terminator is control-L. The last page terminator before the end of file is not physically present in the file, but it is assumed to be present by the rest of the system.

Type COUNT is range 0 .. MAX_INT-1 in TEXT_IO
Type COUNT is range 0 .. MAX_INT in DIRECT_IO

Annex A. Predefined Language Attributes

There are no implementation-defined attributes.

The representation attribute 'address is not supported for packages and labels. An 'address attribute for a subprogram refers to the address of the procedure header for the body of the subprogram. For a task it is the address of the first byte of the task's code. It is only implemented for tasks that are declared through the use of task types. For a constant, 'address yields either the address of a constant or NULL_ADDRESS if no storage location is associated with the constant.

When interfacing to C, FORTRAN, or Pascal, the compilers for these languages force the name in the object code to correspond to the appropriate convention. Thus, for C, the first character of the name has an underscore prepended, FORTRAN has an underscore added in front of and behind the name, and Pascal has an underscore added in front and two underscores added behind the given name.

The following pragmas have no effect:

- CONTROLLED
- MEMORY_SIZE
- OPTIMIZE
- STORAGE_UNIT
- SYSTEM_NAME

The default value for pragma LIST is ON unless the first pragma LIST statement encountered specifies ON, in which case the default is OFF.

The following implementation-dependent pragmas are supported:

COMMENT	Embeds the text of a string literal within the object file of the compilation unit containing the pragma. The syntax is: pragma COMMENT ("text of the comment"); This pragma may appear at any location within the source code of an Ada unit. There is no restriction on the number of comments that may be used.

IMAGES

Controls where the code to support the 'images attribute for an enumeration type is generated. The syntax is:

pragma IMAGES (<enum_name>), DEFERRED | IMMEDIATE);

<enum_name> must be the name of a previously defined enumeration type. This pragma must appear in the same package specification or declarative part as the type definition. If IMMEDIATE is specified, the code for the 'images attribute is generated in the compilation unit where the type definition appears. This is the default. If DEFERRED is specified, the code is generated in any compilation unit that references the 'images attribute. Note that if no references are made to the 'images attribute, no code is ever generated.

LINKNAME

Pragma LINKNAME is used to associate a string with the name of a routine in the object code. The syntax is:

pragma LINKNAME (<ada_name>, <string>);

<ada_name> must be the name of an Ada routine that previously appeared in a pragma INTERFACE. The effect of this pragma is to use the <string> as the name for the routine in the object code for the unit. Thus,

```
function Hyperbolic_Sin (X: Float)
   return Float;
pragma INTERFACE (Assembly, Hyperbolic_Sin);
pragma LINKNAME (Hyperbolic_Sin, ":HSIN:");
```

would cause the compiler to use the string ":HSIN:" in the object code whenever referring to the Ada routine Hyperbolic_Sin.

When interfacing to C, FORTRAN, or Pascal, the compilers for these languages force the name in the object code to correspond to the appropriate Gould convention. The Ada compiler automatically adapts to these conventions. The Ada compiler enforces the following conventions on the names inserted into the object code, where <name> is the subprogram name specified in a pragma INTERFACE statement:

_<name>	C
<name>	FORTRAN
_<name>__	Pascal

Annex C. Predefined Language Environment

Package LOW_LEVEL_IO is not provided.
Package MACHINE_CODE is not provided.

Neither SEQUENTIAL_IO nor DIRECT_IO raises DATA_ERROR.

Type INTEGER is range -(2**31)..(2**31)-1;

Type SHORT_INTEGER is range -(2**15)..(2**15)-1;

Type LONG_INTEGER is range -2**63..(2**63) -1;

Ada FLOAT and LONG_FLOAT use the standard Gould floating point format as described in the CONCEPT 32/67xx and CONCEPT 32/97xx Reference Manuals and the PowerNode 60xx and PowerNode 90xx Reference Manuals.

Type DURATION is delta 1.0/(2.0**14) range -86400.0 .. 86400.0;

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 An identifier the size of the maximum input line length which is identical to $BIG_ID2 except for the last character.	(1..199 => 'A', 200 => '1')
$BIG_ID2 An identifier the size of the maximum input line length which is identical to $BIG_ID1 except for the last character.	(1..199 => 'A', 200 => '2')
$BIG_ID3 An identifier the size of the maximum input line length which is identical to $BIG_ID4 except for a character near the middle.	(1..100 => 'A', 101 => '3', 102..200 => 'A')

TEST PARAMETERS

Name and Meaning	Value
$BIG_ID4 An identifier the size of the maximum input line length which is identical to $BIG_ID3 except for a character near the middle.	(1..100 => 'A', 101 => '4', 102..200 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197 => '0', 198..200 => "298")
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..194 => '0', 195..200 => "69.0E1")
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1 => '"', 2..101 => 'A', 102 => '"')
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1 => '"', 2..100 => 'A', 101 => '1', 102 => '"')
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	(2**63)-2
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	16777215
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8

Name and Meaning	Value
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	GOULD_ADA
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	1.0/(2.0**31)
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	1000
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_FIXED_TYPE
$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NO_SUCH_FLOAT_TYPE
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	10000000.0
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	255
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	(1..257 => 'B')
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	(1..257 => 'A')
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-(2**31)

TEST PARAMETERS

Name and Meaning	Value
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	(2**31)-1
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	(2**31)
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-10000000.0
$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	14
$MAX_IN_LEN Maximum input line length permitted by the implementation.	200
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	(2**63)-1
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	(2**63)
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2 => "2:", 3..197 => '0', 198..200 => "11:")

Name and Meaning	Value
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..3 => "16:", 4..196 => '0', 197..200 => "F.E:")
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1 => '"', 2..199 => 'A', 200 => '"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-(2**63)
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NO_SUCH_TYPE
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	GOULD_ADA
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFFFFFFFFFE#
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	16777215

TEST PARAMETERS

Name and Meaning	Value
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	GOULD_ADA
$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32
$TICK A real literal whose value is SYSTEM.TICK.	1.0/60.0

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 43 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this text that must appear at the top of the page.

b. A39005G: This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

c. B97102E: This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

d. BC3009B: This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

e. CD2A62D: This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

f. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

g. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86, 96, and 58, respectively).

h. CD2B15C and CD7205C: These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

i. CD2D11B: This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

j. CD5007B: This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

k. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

l. CD7105A: This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

m. CD7203B and CD7204B: These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

n. CD7205D: This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

o. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).

p. CE3111C: This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

q. CE3301A: This test contains several calls to END_OF_LINE and END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118,

132, and 136).

r. CE3411B: This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.